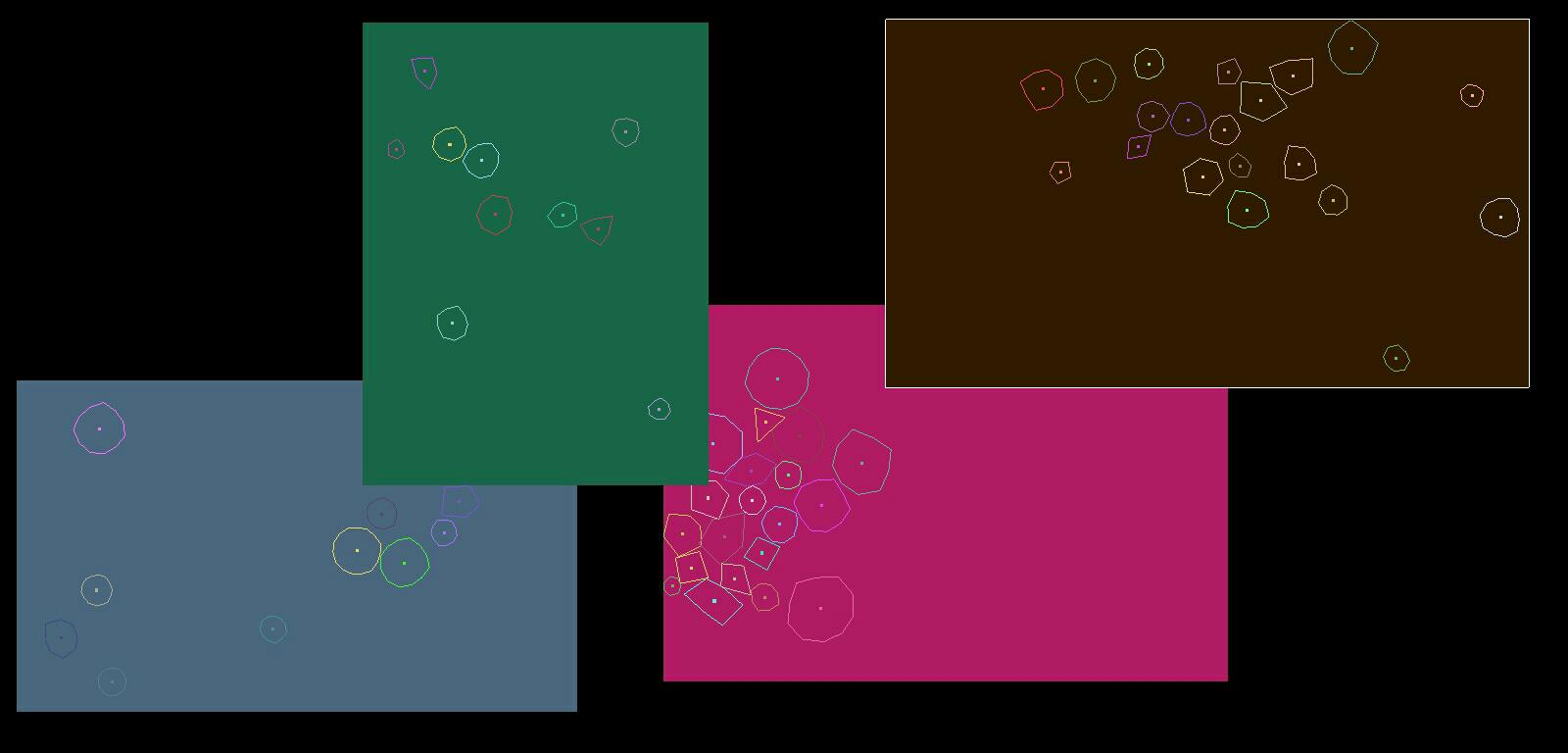
Отчет о проекте

Dynamics

Простой двумерный физический движок



Нечепоренко Степан

Цель:

Написание физического движка для произвольных двумерных многоугольников

Технические требования:

Пользователь задает количество и граничные свойства многоугольников (максимальное и минимальное число сторон и т.д.). Программа генерирует и размещает их так, чтобы они не пересекались. Многоугольники взаимодействуют согласно законам классической физики

Средства:

C++ – быстрый и гибкий, отлично подходит для подобных проектов

freeGLUt – свободная реализация заброшенной GLUt, просто и в тоже время имеет широкий спектр возможностей (из которых в данном проекте были использованы только разноцветные линии и прямоугольники)

Реализация:

Помимо генерации случайных выпуклых многоугольников и симуляции физики были также реализованы следующие возможности:

- Создание и обработка произвольного числа систем – не взаимодействующих между собой множеств тел, заключенных в прямоугольные «стенки»

- Гравитационное взаимодействие многоугольников

- Взаимодействие с пользователем во время симуляции: на определенные в файле keys.txt клавиши назначены такие действия как переключение между системами и их перемещение, изменение длины «стенок», реверс гравитации (отталкивание вместо притягивания), пауза, изменение скоростей на противоположные

Программа состоит из 5-ти файлов

1. Файлы исходного кода: main.cpp, engine.cpp, UI.cpp, keys.cpp

1.1. В файле main.cpp находятся включения необходимых библиотек и функция main, производящая запуск UI, движка и загружающая горячие клавиши

1.2. Файл engine.cpp содержит описание функций, необходимых для рендеринга и вычислений

1.2.1. В классе Vector2 реализован обычный двумерный вектор, но с возможностью несколько модифицированного векторного умножения. Это сделано для удобства реализации вычислительной части, т.к. в исходных формулых присутствует векторное умножение. Для класса перегружены арифметические операции и реализованы методы для подсчета длины, квадрата длины и для поворота на угол a

1.2.2. Класс Color используется для хранения цвета тел и фона системы. Также его можно использовать для генерации случайного цвета (пустой конструктор)

1.2.3. Функция drawLine рисует линию из точки a в точку b, предварительно устанавливая цвет color

1.2.4. Функция frand возвращает случайное число с плавающей запятой на отрезке [a, b]

1.2.5. Класс Body содержит описание тела, являющегося двумерным многоугольником

1.2.5.1. В конструкторе реализован простой метод получения случайного выпуклого многоугольника. Как нетрудно заметить, минимальное расстояние точки от центра, при котором многоугольник гарантированно остается выпуклым равно R\*cos(a), где R - максимальное расстояние точки от центра, a - угол между двумя соседними вершинами многоугольника. После генерации фигуры рассчитывается ее центр масс и вершины сдвигаются таким образом, чтобы вращение происходило вокруг него

1.2.5.2. В методе apply\_impulse после непосредственно применения импульса скорость вращения уменьшается на 10%. Это сделано для симуляции трения и чтобы тела не раскручивались слишком сильно

1.2.6. Функция orient определяет, в какой полуплоскости находится точка c относительно прямой ab

1.2.7. Функции gravitate\_pair и reverse\_gravitate\_pair описывают соответственно гравитационное и антигравитационное взаимодействие двух тел

1.2.8. Класс System описывает систему тел. Система тел здесь - множество выпуклых многоугольников, чье взаимодействие изолировано от прочих тел. Система имеет границы (окна, в рамках которого происходит взаимодействие), цвет фона, имя и собственно список тел.

1.2.8.1. При создании новой системы необходимо задать крайние значения параметров тел, их количество, границы системы и цвет ее фона. Тела размещаются так, чтобы в начальный момент не пересекаться, однако при относительно большом числе/размере тел не предусмотрено предупреждение о недостатке места

1.2.8.2. Метод draw производит рендеринг всей системы

1.2.8.3. Метод move\_bodies двигает тела системы согласно их скоростям и заданому глобально кванту времени

1.2.8.4. Метод solve\_collision предназначен для проверки проникновения вершин тела b1 в тело b2, рассчета и применения к ним корректных импульсов

1.2.8.4.1. В цикле по всем вершинам тела b1 сначала проверяется, находится ли данная вершина в тех же полуплоскостях что и центр тела b2 относительно каждой стороны тела b2. В дальнейшем эта вершина называется contact\_point

1.2.8.4.2. Затем для определения, сквозь какую сторону тела b2 произошло проникновение, находится сторона b2, до которой расстояние от contact\_point минимально. Затем с помощью этих данных считается нормаль контакта contact\_norm (единичный вектор, направленный перпендикулярно к стороне, сквозь которую произошло проникновение)

1.2.8.4.3. Далее производится рассчет вспомогательных величин n1, w1, n2, w2, которые, кстати, являются столбцами матрицы Якоби. Примечание: w1 и w2 имеют тип GLdouble, однако физический их смысл - векторы, имеющие ненулевой только третью координату, которая собственно и является числом

1.2.8.4.4. Т.к. в дальнейших рассчетах координаты центров тел b1 и b2 использоваться не будут, они здесь сдвигаются по нормали контакта так, чтобы развести тела от проникновения до касания (и даже чуть дальше)

1.2.8.4.5. Далее производится рассчет собственно амплитуды импульса лямбда. Рассчет производится исходя из того, что столкновение упругое. Лямбда вычисляется такой, чтобы конечная проекция относительной скорости тел на нормаль контакта была обратна начальной

1.2.8.4.5. Люмбда ограничивается минимальным значением, чтобы столкновения на малых скоростях проходили более жестко. После этого к обоим телам применяются соответствующие импульсы

1.2.8.5. Метод solve\_collisions проверяет все пары тел на возможность столкновения (сумма радиусов больше расстояния между центрами) и реализует отскок тел от стен

1.2.8.6. Метод gravitate реализует гравитационное взаимодействие между телами системы

1.2.8.7. Методы move\_x и move\_y сдвигают окно и тела системы на указанное расстояние

1.3. В файле UI.cpp содержится описание пользовательского интерфейса, реализованного через терминал.

1.3.1. Функции get\_double и get\_int позволяют получить на ввод соответствующее число в заданных пределах при любом поведении пользователя

1.3.2. Функция input аналогична таковой в Python и использует вышеупомянутые функции

1.3.3. Функция setup\_new\_system запрашивает у пользователя параметры новой системы и добавляет ее в список

1.3.4. Функция UI реализует диалог с пользователем

1.4. Файл keys.cpp содержит функции, реализующие возможность взаимодействия с уже созданными системами с помощью горячих клавиш

1.4.1. keys представляет собой массив указателей на функции без аргументов, которые ничего не возвращают. Функция nothing используется для заполнения тех ячеек массива, которые не были заполнены указателями на другие функции

1.4.2. Функция setup\_keys устанавливает настройки горячих клавиш из заданного файла

1.4.3. Функция key устанавливается в main.cpp в качестве обработчика нажатий клавиш и вызывает функцию, соответствующую коду нажатой клавиши

2. Файл со списком горячих клавиш keys.txt